# **Assigned**

# --人为设值

**特点：可以跨数据库，人为控制主键生成，应尽量避免。**

# **Increment**

# --由Hibernate从数据库中取出主键的最大值（每个session只取1次），以该值为基础，每次增量为1

**特点：跨数据库，不适合多进程并发更新数据库，适合单一进程访问数据库，不能用于群集环境。**

# **Hilo**

--hilo（高低位方式high low）是hibernate中最常用的一种生成方式，需要一张额外的表保存hi的值。保存hi值的表至少有一条记录（只与第一条记录有关），否则会出现错误

hilo生成器生成主键的过程（以hibernate\_unique\_key表，next\_hi列为例）：

1. 获得hi值：读取并记录数据库的hibernate\_unique\_key表中next\_hi字段的值，数据库中此字段值加1保存。

2. 获得lo值：从0到max\_lo循环取值，差值为1，当值为max\_lo值时，重新获取hi值，然后lo值继续从0到max\_lo循环。

3. 根据公式 hi \* (max\_lo + 1) + lo计算生成主键值。

注意：当hi值是0的时候，那么第一个值不是0\*(max\_lo+1)+0=0，而是lo跳过0从1开始，直接是1、2、3……

那max\_lo配置多大合适呢？

这要根据具体情况而定，如果系统一般不重启，而且需要用此表建立大量的主键，可以吧max\_lo配置大一点，这样可以减少读取数据表的次数，提高效率；反之，如果服务器经常重启，可以吧max\_lo配置小一点，可以避免每次重启主键之间的间隔太大，造成主键值主键不连贯。

**特点：跨数据库，hilo算法生成的标志只能在一个数据库中保证唯一。**

# **4、seqhilo**

--与hilo类似，通过hi/lo算法实现的主键生成机制，只是将hilo中的数据表换成了序列sequence，需要数据库中先创建sequence，适用于支持sequence的数据库，如Oracle。

**特点：与hilo类似，只能在支持序列的数据库中使用。**

# **5、sequence**

**--**采用数据库提供的sequence机制生成主键，需要数据库支持sequence。如oralce、DB、SAP DB、PostgerSQL、McKoi中的sequence。MySQL这种不支持sequence的数据库则不行（可以使用identity）。

**特点：只能在支持序列的数据库中使用，如Oracle。**

# **6、identity**

--identity由底层数据库生成标识符。identity是由数据库自己生成的，但这个主键必须设置为自增长，使用identity的前提条件是底层数据库支持自动增长字段类型，如DB2、SQL Server、MySQL、Sybase和HypersonicSQL等，Oracle这类没有自增字段的则不支持。

**特点：只能用在支持自动增长的字段数据库中使用，如MySQL。**

**自动增长字段类型与序列**

|  |  |  |
| --- | --- | --- |
| **数据库** | **自动增长字段** | **序列** |
| **MySQL** | **是** |  |
| **Oracle** |  | **是** |
| **DB2** | **是** | **是** |
| **MS SQL Server** | **是** |  |
| **Sybase** | **是** |  |
| **HypersonicSQL** | **是** |  |
| **PostgreSQL** |  | **是** |
| **SAP DB** |  | **是** |
| **HSQLDB** | **是** |  |
| **Infomix** | **是** |  |

# **7、native**

--native由hibernate根据使用的数据库自行判断采用identity、hilo、sequence其中一种作为主键生成方式，灵活性很强。如果能支持identity则使用identity，如果支持sequence则使用sequence。

**特点：根据数据库自动选择，项目中如果用到多个数据库时，可以使用这种方式，使用时需要设置表的自增字段或建立序列，建立表等。**

# **8、uuid**

**--**UUID：Universally Unique Identifier，是指在一台机器上生成的数字，它保证对在同一时空中的所有机器都是唯一的。

**特点：uuid长度大，占用空间大，跨数据库，不用访问数据库就生成主键值，所以效率高且能保证唯一性，移植非常方便，推荐使用。**

# **9、guid**

--GUID：Globally Unique Identifier全球唯一标识符，也称作 UUID，是一个128位长的数字，用16进制表示。

**注意：长度因数据库不同而不同**

MySQL中使用select uuid()语句获得的为36位（包含标准格式的“-”）

Oracle中，使用select rawtohex(sys\_guid()) from dual语句获得的为32位（不包含“-”）

**特点：需要数据库支持查询uuid，生成时需要查询数据库，效率没有uuid高，推荐使用uuid。**

# **10、foreign**

--使用另外一个相关联的对象的主键作为该对象主键。主要用于一对一关系中

**特点：很少使用，大多用在一对一关系中。**

# **11、select**

**--**使用触发器生成主键，主要用于早期的数据库主键生成机制，能用到的地方非常少。

# **自定主键生成**

package com.fwd.eprecious.util;

import java.io.Serializable;

import java.util.List;

import java.util.Properties;

import org.hibernate.HibernateException;

import org.hibernate.MappingException;

import org.hibernate.dialect.Dialect;

import org.hibernate.engine.spi.QueryParameters;

import org.hibernate.engine.spi.SessionImplementor;

import org.hibernate.id.AbstractUUIDGenerator;

import org.hibernate.id.Configurable;

import org.hibernate.id.IdentifierGenerator;

import org.hibernate.id.UUIDGenerationStrategy;

import org.hibernate.type.Type;

public class GeneratePK implements Configurable , IdentifierGenerator{//, IdentifierGenerator extends AbstractUUIDGenerator

public String sign;// user000000001中的user

public String classname; // 实体类的类名

public String pk;// 主键名字

public String idLength;// user000000001的长度

@Override

public void configure(Type type, Properties params, Dialect d) throws MappingException {

this.classname = params.getProperty("classname");

this.pk = params.getProperty("pk");

this.sign = params.getProperty("sign");

this.idLength = params.getProperty("idLength");

}

@Override

public Serializable generate(SessionImplementor session, Object object) throws HibernateException {

//获得主键的长度

int leng = Integer.valueOf(idLength);

//需要查询数据库中最大的ID号

StringBuffer sql = new StringBuffer("select max(a.").append(pk)

.append(") from ")

.append(classname)

.append(" as a where a.")

.append(pk)

.append(" like '")

.append(sign)

.append("%'");

QueryParameters qp = new QueryParameters();

List ls = session.list(sql.toString(), qp);

String max = (String) ls.get(0);

int i = 0;

//如果是第一次添加记录那么就是类似user000000001

if (max == null || max.trim().equals("")) {

max = "1";

for(; i < leng-sign.length()-1; i++) {

max = "0" + max;

}

i = 0;

return sign + max;

}//不是第一次的操作,并且记录的长度没有超过从配置文件中读取的长度

else if(max != null && max.length() <= leng) {

max = max.replaceAll(sign, "");

Integer imax = Integer.parseInt(max) + 1;

String returnnum = String.valueOf(imax);

int zero = leng-sign.length()-returnnum.length();

for(; i < zero; i++) {

returnnum = "0" + returnnum;

}

i = 0;

return sign + returnnum;

}//不是第一次的操作,记录的长度超过了从配置文件中读取的长度

else {

leng = max.length();

max = max.replaceAll(sign, "");

Integer imax = Integer.parseInt(max) + 1;

String returnnum = String.valueOf(imax);

int zero = leng-sign.length()-returnnum.length();

for(; i < zero; i++) {

returnnum = "0" + returnnum;

}

return sign + returnnum;

}

}

}

@Id

@GeneratedValue(generator = "idGenerator")

@GenericGenerator(name = "idGenerator", strategy = "com.fwd.eprecious.util.GeneratePK",

parameters = { @Parameter(name = "sign", value = "ag") ,@Parameter(name = "classname", value = "Agent")

,@Parameter(name = "pk", value = "agentseq"),@Parameter(name = "idLength", value = "11")})

**private** String agentseq;